PARTIAL ADDRESS COMPARES STORED IN TRANSLATION LOOKASIDE BUFFER

Michael D. Estlick

Harry R. Fair, III

5   David R. Akeson

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to the field of processors and more particularly to translation lookaside buffers within processors.

10   Description of the Related Art

In computer systems it is known for a processor to have a cache memory to speed up memory access operations to main memory of the computer system. The cache memory is smaller, but faster than main memory. It is placed operationally between the processor and main memory. During the execution of a software

15   program, the cache memory stores more frequently used instructions and data. Whenever the processor needs to access information from main memory, the processor examines the cache first before accessing main memory. A cache miss occurs if the processor cannot find instructions or data in the cache memory and is required to access the slower main memory. Thus, the cache memory reduces the

20   average memory access time of the processor.

In known computer systems, it is common to have a process executing only in main memory ("physical memory") while a programmer or user perceives a much larger memory which is allocated on an external disk ("virtual memory"). Virtual memory allows for very effective multi-programming and relieves the user of

25   potential constraints associated with the main memory. To address the virtual

memory, many processors contain a translator to translate virtual addresses in virtual memory to physical addresses in physical memory, and a translation lookaside buffer ("TLB"), which caches recently generated virtual-physical address pairs. The TLBs allow faster access to main memory by skipping the mapping process when the

5   translation pairs already exist. A TLB entry is like a cache entry where a tag includes portions of the virtual address and a data portion includes a physical page frame number.

One aspect of processor performance relates to monitoring certain addresses such as instruction addresses via, for example, a watchpoint address or a sample

10   address range. When monitoring the instruction address, it becomes important to quickly compare the instruction address against the watchpoint address or the sample address range. What a match is detected between the instruction address and the monitoring address, the processor takes some sort of action such as generating a watchpoint trap if the address matches the watchpoint address or collecting sampling

15   information if the instruction address is within the sample address range.

SUMMARY OF THE INVENTION

In accordance with the present invention, a method for performing a fast information compare within a processor is set forth in which a more significant bit

20   compare is performed when information is being loaded into a translation lookaside buffer. The result of the more significant bit compare is stored within the translation lookaside buffer as part of an entry containing the information. When the fast compare is desired, the result of the more significant bit compare is used in conjunction with results from a compare of less significant bits of the information and

25   less significant bits of a compare address to determine whether a match is present.

In one embodiment, the invention relates to a method of performing a fast information compare within a processor which includes performing a more significant bit compare when information is loaded into a translation lookaside buffer, storing a result of the more significant bit compare within the translation lookaside buffer as

part of an entry containing the information, and using the result of the more significant bit compare in conjunction with results from a compare of less significant bits of the information and less significant bits of compare information to determine whether a match is present. The more significant bit compare compares more

5   significant bits of the information being loaded into the translation lookaside buffer with more significant bits of compare information.

In another embodiment, the invention relates to an apparatus for performing a fast information compare within a processor which includes means for performing a more significant bit compare when information is loaded into a translation lookaside

10   buffer, means for storing a result of the more significant bit compare within the translation lookaside buffer as part of an entry containing the information, and means for using the result of the more significant bit compare in conjunction with results from a compare of less significant bits of the information and less significant bits of compare information to determine whether a match is present. The more significant

15   bit compare compares more significant bits of the information being loaded into the translation lookaside buffer with more significant bits of compare information;

In another embodiment, the invention relates to a processor which includes a translation lookaside buffer, a first compare unit coupled to the translation lookaside buffer and a second compare unit coupled to the translation lookaside buffer. The

20   first compare unit performs a more significant bit compare when information is loaded into a translation lookaside buffer. The more significant bit compare compares more significant bits of the information being loaded into the translation lookaside buffer with more significant bits of compare information. The first compare unit stores a result of the more significant bit compare within the translation lookaside

25   buffer as part of an entry containing the information. The second compare unit processor uses the result of the more significant bit compare in conjunction with results from a compare of less significant bits of the information and less significant bits of compare information to determine whether a match is present.

In another embodiment, the invention relates to a processor which includes a

30   memory management unit and an instruction fetch unit. The memory management unit includes a memory management unit translation lookaside buffer. The

-3-

instruction fetch unit includes an instruction translation lookaside buffer. The more significant bit compare is performed when information is loaded into the instruction translation lookaside buffer.

5   **BRIEF DESCRIPTION OF THE DRAWINGS**

The present invention may be better understood, and its numerous objects, features and advantages made apparent to those skilled in the art by referencing the accompanying drawings. The use of the same reference number throughout the several figures designates a like or similar element.

10   Figure 1 shows a schematic block diagram of a processor architecture.

Figure 2 shows a schematic block diagram of the interaction of a memory management unit with other portions of a processor.

Figure 3 shows a block diagram of a virtual address translation.

Figure 4 shows a block diagram of a micro translation look aside buffer.

15   Figure 5 shows a block diagram of a micro translation look aside buffer entry.

Figure 6 shows a block diagram of the operation of portions of the processor when performing a fast address compare.

DETAILED DESCRIPTION

Figure 1 shows a schematic block diagram of a multithreaded processor
20   architecture. More specifically, processor 100 includes an instruction fetch unit (IFU) 110, an instruction renaming unit (IRU) 112, an instruction scheduling unit (ISU) 114, a floating point and graphics unit (FGU) 120, an integer execution unit (IEU) 122, a memory management unit (MMU) 130, a data cache unit (DCU) 132, a secondary cache unit (SCU) 140, an external interface unit (EIU) 142. The processor also

includes a test processing unit (TPU) 150 and a performance hardware unit (PHU) 152.

The instruction fetch unit 110 includes an instruction cache and branch prediction logic. The instruction fetch unit 110 is coupled to the instruction renaming unit as well as to the memory management unit 130 and the secondary cache unit 140.

The instruction renaming unit is coupled to the instruction fetch unit 110 and to the instruction scheduling unit 114. The instruction renaming unit 112 includes dependency check logic and a helper instruction generator.

The instruction scheduling unit is coupled to the floating point and graphics unit 120 and to the integer execution unit 122. The instruction scheduling unit 114 includes an instruction window module.

The floating point and graphics unit 120 is coupled to the instruction scheduling unit 114 and to the floating point and data cache unit 132. The floating point and graphics scheduling unit 120 includes floating point and graphics execution units, a floating point register file and a floating point and graphics result buffer.

The integer execution unit 122 is coupled to the instruction scheduling unit 114 and to the data cache unit 132. The integer execution unit 122 includes integer execution units, an integer register file and virtual address adders.

The memory management unit 130 is coupled to the instruction fetch unit 110 and to the secondary cache unit 140. The memory management unit 130 includes a virtual address to physical address translation module as well as a translation lookaside buffer.

The data cache unit 132 is coupled to the floating point and graphics unit 120, to the integer execution unit 122 and to the secondary cache unit 140. The data cache unit 132 includes a data cache and a memory disambiguation buffer.

The secondary cache unit 140 is coupled to the memory management unit 130, the data cache unit 132 and the external interface unit 142. The secondary cache unit 140 includes a memory scheduling window as well as a unified L2 (level 2) cache.

The external interface unit 142 is coupled to the secondary cache unit 140 as well as to an external cache and an input/output (I/0) controller. The external interface unit 142 includes a transaction scheduling window, an external cache controller and an I/O system interconnection controller.

5   The test processing unit 150 is coupled to various units across the processor 100. The test processing unit 150 includes a power on controller as well as a clock controller.

The performance hardware unit 152 is coupled to various units across the processor 100. The performance hardware unit includes performance instrumentation
10   counters as well as a sampling mechanism.

The instruction fetch unit 110 is responsible for fetching instructions from the instruction cache and then sending the resulting bundles of instructions to the instruction renaming unit 112. The instruction fetch unit may fetch up to eight instructions per cycle. Each group of instruction s delivered to by the instruction
15   fetch unit is referred to as a fetch bundle. The instruction cache sources instructions to the processor pipeline by accessing a local instruction cache with predetermined cache indices. The instruction is virtually addressed by an instruction pointer generator. The branch prediction logic enables the instruction fetch unit 110 to speculatively fetch instruction s beyond a control transfer instruction (CTI) even
20   though the outcome or target of the control transfer instruction is not yet known.

The instruction renaming unit 112 decodes instructions, determines instruction dependencies and manages certain processor resources. The instruction scheduling unit 114 scheduling instructions from each thread for execution, replays instructions that are consumers of loads when the load misses in the data cache, maintains
25   completion and trap status for instructions executing within the processor 100 and separately retires instruction sin fetch order from each thread.

The floating point execution unit 120 implements and executes floating point instructions and graphics instructions. The integer execution unit 122 implements and executes fixed point integer instructions. Additionally, the integer execution unit 122

assists in execution of floating point instruction which depend on integer condition codes, integer registers and floating point condition codes.

The memory management unit 130 performs virtual address to physical address translation and includes a translation lookaside buffer that provides fro a

5   translation for the most frequently accessed virtual pages.

The data cache unit 132 provides the main interface between execution pipelines and memory within the processor 100. The data cache unit 132 executes load and store instructions as well as derivatives of load and store instructions. The data cache unit 132 provides a first level cache that is coupled directly to the

10   execution units. The memory disambiguation buffer dynamically disambiguates memory addresses to enable execution of out of order instructions.

The secondary cache unit 140 provides a unified L2 cache. The L2 cache is controlled by the memory scheduling window which tracks accesses that miss in the L1 caches, the MMU and snoop system request. The memory scheduling window

15   provides an interface between the instruction fetch unit and the L2 cache. The memory scheduling window also receives snoops from the external interface unit 142 and retired stores from the data cache unit 132.

The external interface unit 142 controls data flow among the L2 cache and the external cache, controls system interconnect, provides external cache control and

20   provides a common interface for external processors, I/O bridges, graphics devices, and memory controllers.

The test processing unit 150 performs power on tests as well as diagnostic access within the processor 100. The test processing unit 150 provides clock control, design for testability and access to external interfaces.

25   The performance hardware unit 152 uses the performance instrumentation counters to gather aggregate information about various performance events across a plurality of instructions. The sampling mechanism gathers more detailed instruction history for specific executions of a sampled instruction.

Referring to Figure 2, a schematic block diagram setting forth the interaction of a memory management unit with other portions of the processor 100 is shown. More specifically, the memory management unit 130 caches address mappings. Programs operate in a virtual address space. The memory management unit 130

5   translates virtual addresses that a program uses into the physical addresses of where the information actually resides. By making a distinction between the address used to reference data and the address where the data resides, an operating system may provide each program with its own address space and may enforce access permissions.

10   The operating system assigns each address space an identifying number (a context) and dividing the memory space into pages. Translation is performed by keeping virtual address bits which are a page offset and replacing the rest of the virtual address with a physical address. Each page has a virtual address, a physical address, and a context as well as attribute bits which determine how a program may

15   access the page. A mapping is the association of the virtual address and context to the physical address. The memory management unit 130 provides a physical address when provided a virtual address and a context. The memory management unit 130 also enforces how the data may be accessed.

The operating system maintains a list of virtual to physical address mappings.

20   The memory management unit 130 speeds up the translation process by storing commonly used mappings within a translation lookaside buffer (TLB). The memory management unit 130 adds new mapping when needed and evicts no longer needed mappings. When a request to the memory management unit 130 misses, indicating that the memory management unit does not have a requested mapping, the memory

25   management unit 130 queries the operating system maintained list to serve the request.

The processor 100 includes two levels of memory mapping caching. The first level of caching is within an instruction TLB located within the instruction fetch unit 110 for instruction mappings and within a data TLB located within the data cache unit

30   132 for data mappings. When either the instruction TLB or the data TLB miss, then

the missing TLB makes a request to the second level TLB stored within the memory management unit 130.

In one embodiment, the memory management unit includes two TLBs, a 2048 entry, 4 way set associated structure and a 32 entry content addressable memory
5   structure. The memory management unit 130 maps a 64 bit virtual address space onto a 47 bit physical address space.

The data TLB supports access permission s for data accesses, while the memory management unit supports instruction accesses. The memory management unit supports access to a translation storage buffer, which is a direct mapped structure
10   in memory which holds memory mappings as translation table entries. The memory management unit may either directly query the translation storage buffer via hardware or may generate a trap which allows software to query the translation storage buffer and then write the mapping into the memory management unit when an access causes the memory management unit misses on a mapping.

15   Figure 3 shows a block diagram of a virtual address translation. The size of the virtual page number, physical page number and page offset depends on the page size. For example, for an 8K page size, X equals 13, for a 64K page size, X equals 16 and for a 512K page size, X equals 19. Other page sizes, such as 4M (Megabyte), 32M, 256M, 2G (Gigabyte), and 16G page sizes may also be used.

20   Figure 4 shows a block diagram of an instruction micro translation lookaside buffer module 400. The instruction micro translation lookaside buffer module 400 includes a virtual page content addressable memory (CAM) (VPC) 410, an instruction translation lookaside buffer portion (ITB) 412, and a virtual page array (VPA) 414. The instruction micro translation lookaside buffer module 400 interacts with the
25   memory management unit 130 as well as a branch address calculator (BAC) module 430, a branch repair table (BRT) 432 and an instruction address queue (IAQ) module 434, each of which are located within the instruction fetch unit 110.

The instruction micro translation look aside buffer module 400 performs first level virtual to physical address translations. The virtual page CAM 410 functions as
30   a tag portion of the array and the instruction translation lookaside buffer portion 412

functions as a data portion of the array. The virtual page array 414 provides a direct mapped index predictor into the instruction translation lookaside buffer portion 412.

In operation, during a fast and common case of address translation, the virtual page array 414 predicts the index of the correct entry in the instruction translation
5   lookaside buffer portion 412. The instruction translation lookaside buffer portions 412 provides an output of both a virtual page number (vpn) and a physical page number (ppn) of the translation so that the prediction can be verified.

In the case of a branch address calculator mispredict or a branch repair table redirect, the correct program count is stored within the virtual page CAM. The virtual
10   page CAM provides a virtual page index (vpi) into the instruction translation lookaside buffer module 412. The virtual page index of the virtual page CAM 410 is also used to train the virtual page array 414. If the translation does not reside within the micro translation look aside buffer module 400, then the virtual page CAM initiates a request for a translation to the memory management unit 130.

15   The memory management unit 130 either provides the translation to the instruction micro translation look aside buffer 400 or generates a MMU trap to indicate that the MMU does not have the translation stored within the second level TLB. When receiving the translation from the memory management unit, the virtual page CAM 410 and the instruction translation lookaside buffer module 412 are
20   updated.

Figure 5 shows a block diagram of a micro translation look aside buffer entry. More specifically, each entry of the instruction TLB includes a mapping from the upper bits of the Virtual Address to the upper bits of the Physical Address. Each entry of the instruction TLB also includes a partial address compare field for the entry. The
25   partial address compare field includes eight bits that represent the partial compare of the upper bits of the Virtual Address to a virtual address watchpoint trap address as well bits that represent whether the address is within a sample address range.

The eight bits include PartialCompareBit[O] through PartialCompareBit [7]. PartialCompareBit[0] represents when the entry has an address between the thread 0
30   sample selection criteria low address and the sample selection criteria high address.

PartialCompareBit[1] represents when the entry has an address below the thread 0 sample selection criteria low address. PartialCompareBit [2] represents when the entry has an address above the thread 0 sample selection criteria high address. PartialCompareBit [3] represents when the entry has an address between the thread 1

5   sample selection criteria low address and the sample selection criteria high address. PartialCompareBit[4] represents when the entry has an address below the thread 1 sample selection criteria low address. PartialCompareBit [5] represents when the entry has an address above the thread 1 sample selection criteria high address. PartialCompareBit [6] represents when the entry has an address which corresponds to

10   the thread 0 watchpoint address. PartialCompareBit [7] represents when the entry has an address which corresponds to the thread 1 watchpoint address.

Because the processor 100 includes two threads, there are bits corresponding to each of the threads. It will be appreciated that processors having other numbers of threads might have a partial address compare bits corresponding to each thread.

15   Referring to Figure 6, a block diagram of the interaction of various processor elements is shown. More specifically, when the translation for a page is written into the instruction micro Translation Lookaside Buffer (ITLB) 400, the partial address compare field is written into the entry to support two address compares and two address range compares. The partial address compare field bits are generated based

20   upon a comparison that is performed by compare unit 605 at the time the TLB entry is stored.

In one embodiment, the processor 100 includes two hardware threads where each thread includes an Instruction Virtual Address Watchpoint (IVAWP) and a Sampling Selection Criteria PC Range (SSC PC Range). The IVAWP is monitored

25   via an address compare, and the SSC PC Range is monitored via an address range compare. There are three bits per address range compare and one bit per address compare.

When performing the address range compare, if the bottom of the address range is A, the top of the address range is B, and the address to compare is X, the

30   three partial compare bits of the SSC PC Range correspond to a sample selection

criteria mid address bit (A <= X < B), a sample selection criteria low address bit (X == A), and a sample selection criteria high address bit (X == B). Three bits are used to perform the address range compare from the upper bits because there are five possible cases to encode. The range is entirely inside the page, the page is entirely

5   inside the range, the top of the range is in the page, the bottom of the range is in the page, and the page is entirely outside of the range.

When instructions are fetched during instruction execution, the ITLB 400 is accessed to obtain the virtual address to physical address mapping. The eight compare bits are also read and used to compute the final address range compares and address

10   compares via compare unit 610. The processor 100 may fetch up to eight instructions in a bundle per cycle.

The IVAWP address compare is an exact address compare and the result is a mask that picks zero or one of the eight instructions in a bundle. The IVAWP is a debug feature that is used to cause a trap to occur on a specific instruction.

15   The SSC PC Range is used to constrain instruction sampling to a range of addresses. The compare is not exact and only determines if any instruction in a bundle is within the SSC PC Range. The SSC PC Range enables sampling on a bundle, and then any instruction inside that bundle might get chosen as a sample.

Because the upper bits of the compares are read from the ITLB, only the lower

20   bits of the address need to be compared by compare unit 610 at fetch time. The work to do the address compare is split between the ITLB fill time and the fetch time. Because the time at which the ITLB is filled is not critical to the performance of the processor 100, there is more time to perform compares at ITLB fill time. Additionally, the results of the compares are cached in the ITLB 400 and can be used

25   many times during the execution of the processor 100.

Other Embodiments

The present invention is well adapted to attain the advantages mentioned as well as others inherent therein. While the present invention has been depicted, described, and is defined by reference to particular embodiments of the invention,

such references do not imply a limitation on the invention, and no such limitation is to be inferred. The invention is capable of considerable modification, alteration, and equivalents in form and function, as will occur to those ordinarily skilled in the pertinent arts. The depicted and described embodiments are examples only, and are

5   not exhaustive of the scope of the invention.

For example, while a particular processor architecture is set forth, it will be appreciated that variations within the processor architecture are within the scope of the present invention.

Also for example, while the partial compare bits are described stored within

10   the instruction translation lookaside buffer, it will be appreciated that the partial compare information may be stored within any translation lookaside buffer of a processor or within other temporary storage units of a processor such that the partial address compare is performed outside of any critical timing paths.

Also for example, the above-discussed embodiments include modules and

15   units that perform certain tasks. The modules and units discussed herein may include hardware modules or software modules. The hardware modules may be implemented within custom circuitry or via some form of programmable logic device. The software modules may include script, batch, or other executable files. The modules may be stored on a machine-readable or computer-readable storage medium such as a

20   disk drive. Storage devices used for storing software modules in accordance with an embodiment of the invention may be magnetic floppy disks, hard disks, or optical discs such as CD-ROMs or CD-Rs, for example. A storage device used for storing firmware or hardware modules in accordance with an embodiment of the invention may also include a semiconductor-based memory, which may be permanently,

25   removably or remotely coupled to a microprocessor/memory system. Thus, the modules may be stored within a computer system memory to configure the computer system to perform the functions of the module. Other new and various types of computer-readable storage media may be used to store the modules discussed herein. Additionally, those skilled in the art will recognize that the separation of functionality

30   into modules and units is for illustrative purposes. Alternative embodiments may merge the functionality of multiple modules or units into a single module or unit or

-14-

may impose an alternate decomposition of functionality of modules or units. For example, a software module for calling sub-modules may be decomposed so that each sub-module performs its function and passes control directly to another sub-module.

Consequently, the invention is intended to be limited only by the spirit and

5   scope of the appended claims, giving full cognizance to equivalents in all respects.